

# LA-UR-13-22909

Approved for public release; distribution is unlimited.

Title: Computational Co-design of a Multiscale Plasma Application: A Process and Initial Results

Author(s): Payne, Joshua E.  
Knoll, Dana A.  
McPherson, Allen L.  
Taitano, William  
Pakin, Scott D.  
Chen, Guangye

Intended for: IPDPS 2014, 2014-05-19/2014-05-23 (Phoenix, Colorado, United States)

Issued: 2014-05-21 (rev.3)



## Disclaimer:

Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the Los Alamos National Security, LLC for the National Nuclear Security Administration of the U.S. Department of Energy under contract DE-AC52-06NA25396. By approving this article, the publisher recognizes that the U.S. Government retains nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

# Computational Co-Design of a Multiscale Plasma Application: A Process and Initial Results

Joshua Payne, Dana Knoll, Allen McPherson, William Taitano, Luis Chacón, Guangye Chen, and Scott Pakin  
*Los Alamos National Laboratory*

**Abstract**—As computer architectures become increasingly heterogeneous the need for algorithms and applications that can exploit these new architectures grows more pressing. This paper demonstrates that co-designing a multi-architecture, multi-scale, highly optimized framework with its associated plasma-physics application can provide both portability across CPUs and accelerators and high performance. Our framework utilizes multiple abstraction layers in order to maximize code reuse between architectures while providing low-level abstractions to incorporate architecture-specific optimizations such as vectorization or hardware fused multiply-add. We describe a co-design process used to enable a plasma physics application to scale well to large systems while also improving on both the accuracy and speed of the simulations. Optimized multi-core results will be presented to demonstrate ability to isolate large amounts of computational work with minimal communication.<sup>1</sup>

**Keywords**—particle-in-cell; co-design; implicit PIC; plasma physics

## I. INTRODUCTION

With the introduction of accelerators such as GPUs and Intel Xeon Phi (MIC) co-processors, computer systems have added multiple levels of parallelism. Multi-node, multi-core systems have roughly three levels of parallelism: inter-node, inter-core, and intra-core (SIMD). Each level has a different amount of memory per parallel quanta and different communication costs. Accelerators add at least one more level of parallelism, and branch the hierarchy as illustrated in Figure 1. These heterogeneous systems now have multiple parallel branches, with each branch having its own parallel hierarchy. Some branches, such as the multi-core branch, have a few “heavy” threads, i.e. threads that require a lot of overhead to create, but can handle a large amount of work. Threads on the GPU are considered “light” in that the overhead for spawning new threads is very low, but each thread is expected to perform only a small portion of the total work.

The tendency towards hierarchical architectures necessitates the development of hierarchical algorithms that map well to these systems. Multi-scale physics problems exhibit a similar hierarchical nature, and therefore may be able to naturally take advantage of both hierarchical architectures and algorithms. Compounding these issues is the difficulty

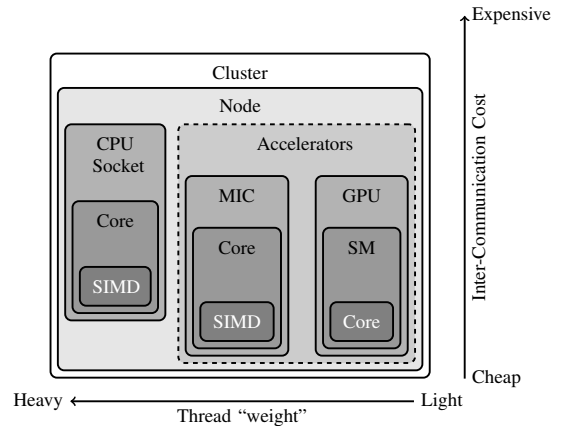


Figure 1. Multiple levels of parallelism in emerging heterogeneous architectures.

of achieving both performance and portability across architectures in the general case. Instead of solving these issues for the general case, we propose a co-design process to be used with specific applications that incorporate underlying, application-specific, abstraction layers to achieve both portability and performance. Our goal is to produce a comprehensive co-design process that can be used to identify applications, develop algorithms, map those algorithms to existing architectures, and optimize emerging architectures to better suit the needs of the science problems.

Essentially, applications and algorithms must adapt together to new architectures while programming models and abstractions must adapt to new applications and algorithms. A co-design effort seeks to achieve an optimized interaction between applications, algorithms, programming models, abstractions, and emerging architectures in order to enable the exploration of new scientific problems. Computational co-design efforts, such as the Computational Co-Design for Multi-Scale Application in the Natural Sciences (CoCoMANS) project at LANL, encourage forming, organizing, and managing multi-disciplinary teams working towards a common goal. The end-goal of our evolving co-design process is to produce a paradigm shift in the pursuit of multi-scale science simulations. This paradigm shift comprises three aspects: 1) demonstrating an effective use of hierarchical parallelism on emerging architectures,

<sup>1</sup>LA-UR-13-22909

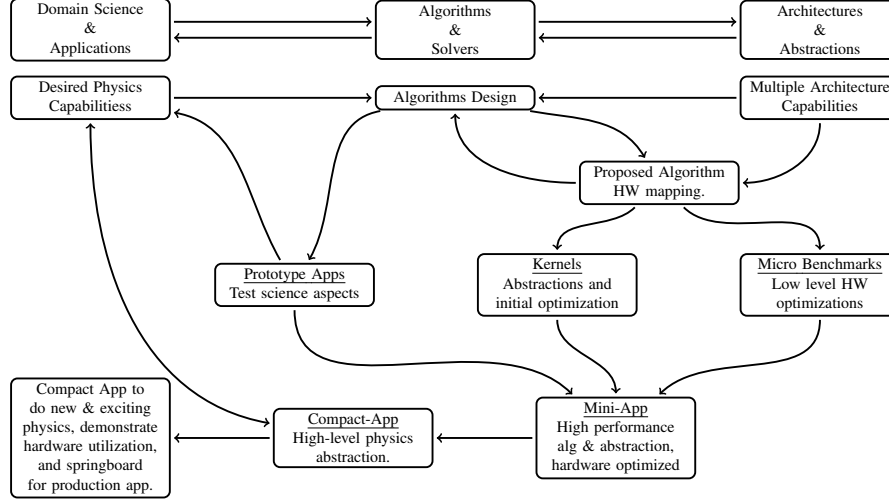


Figure 2. The evolving co-design process used to create a multi-scale plasma physics application. The process begins with identification of desired physics capabilities and architecture capabilities. The algorithm design is influenced by these two factors, and once an algorithm design has been completed it is tested for both scientific and hardware mapping characteristics. The desired physics characteristics and algorithm design are iterated on based on these tests. Eventually the scientific and architectural aspects combine with high-level physics abstractions in a compact-app that is used to study newly accessible physical phenomena.

2) achieving a significant new physics simulation capability via enhanced model fidelity, increased numerical accuracy, or both, and 3) running on multiple types of hardware (GPUs vs. MIC vs. multi-core) with no physics or numerics code changes [11]. The process being evolved, graphically represented in Figure 2 involves an iteration between three primary areas: algorithms, abstractions, and applications.

#### A. Numerical Algorithms and Solvers

Modern computer architectures are becoming increasingly heterogeneous. New supercomputers, such as Titan at ORNL, Tianhe-2 at NUDT, and Stampede at TACC, can attribute a very large portion of their peak FLOPS to accelerators such as GPUs and MIC co-processors. These new architectures promise significant gains in both raw performance and performance per watt. However, the development of algorithms that can take full advantage of these new systems has proven difficult. Algorithms tailored for these heterogeneous systems must; 1) utilize very fine-grained parallelism, 2) isolate a very large amount of work on the accelerators, 3) minimize communication between accelerators and the host system, 4) isolate a very large amount of work on-node, and 5) maximize use of SIMD and SIMT features.

The application outlined in this paper utilizes a particle-based method that maps very well to the fine grained parallelism utilized by accelerators such as GPUs and MIC co-processors. However, cross-accelerator and accelerator-host communication is very expensive, and these communication costs can become significant for problems that involve disparate temporal and spatial scales. For such stiff problems, scale-bridging acceleration techniques are utilized

in order to isolate work on the accelerators and minimize communication.

Such scale-bridging algorithmic acceleration techniques couple low-order (LO), coarse-grained models with high-order (HO), fine-grained descriptions, in order to resolve macroscopic effects based on microscopic behaviors. It possesses a hierarchical nature similar to that of emerging architectures by construction. High-order descriptions alone, such as particle methods for kinetic problems [9], tend to have a very high degree of isolated parallelism. However, these associated algorithms often require frequent global communication in order to ensure that nonlinear macroscopic features are captured. The frequency of communications for each nonlinear iteration can be significantly reduced by using a LO model to aid in the resolution of the macroscopic features.

Our insight therefore is to use a moment-based low-order (LO) method to accelerate the non-linear convergence of a particle based high-order (HO) method. For the specific example of interest in this document, the LO moment-based method simply solves the discretized fluid equations (obtained by a rigorous moment integration), coupled with the HO Vlasov equation (using particles) and the Darwin (non-radiative) formulation of Maxwell's equations. The coupled HO-LO Darwin-PIC formulation is solved implicitly.

Our formulation/algorithmic choices are motivated by the target problems and architectures. Implicit PIC was chosen because it is free of the numerical stability constraints of explicit PIC [2], it allows particle subcycling, and features exact charge and energy conservation [5], [16]. We use a Darwin formulation of Maxwell's equations in order to avoid radiative noise due to numerical light wave dispersion. As

a result of these choices, our approach features a unique combination of algorithmic performance, mapping to architectures, and long-term accuracy. These methods will be explained in more detail later in this document.

### B. Computer Science and Hardware

In addition to the algorithmic challenges, heterogeneous architectures present significant development challenges. The different processing components of these architectures can require substantially different data storage, data management, and execution treatments. An example of this can be seen in structures of arrays on GPUs vs. arrays of structures on CPUs. Often, this leads developers to write multiple versions of the same physics algorithms, in some cases completely rewriting the application in order to run well on different architectures.

This should not be the case, as often the underlying physics (as well as much of the higher level numerical methods) will be the same on all architectures. Language extensions such as OpenCL [13] facilitate this process by relying on the compiler to generate code that can run on a large range of hardware. However, the compiler can not automatically generate code optimized for multiple different pieces of hardware from a single OpenCL kernel. Many device-specific optimizations are based on maximizing cache usage, ensuring that certain intrinsic operations are used, and structuring an algorithm such that it maps well to a specific architecture.

In order to get good code reuse, portability, and performance, multiple layers of abstraction should be identified such that hardware-specific data management and execution control can be hidden from the physics developer. Low-level compile-time controls, such as preprocessor statements, can be combined with higher level programming language concepts such as template meta-programming and virtual classes to direct the compiler. Given the proper direction, the compiler will generate multiple, optimized, versions of the same physics operators for different sets of hardware. Our contribution is an abstraction layer for a kinetic plasma simulation application that utilizes an algorithmic structure matched to the hardware structure by using a HO/LO approach instead of retrofitting an existing algorithm.

### C. Domain Science

Our plasma application area spans multiple spatial and temporal scales, separated by several orders of magnitude, as shown in Figure 3. In the plasma application, we are solving Vlasov equations for ions and electrons along with Maxwell's equations for the electromagnetic fields [2]. The Vlasov equation can have up to three spatial dimensions (3D) and up to three velocity space dimensions (3V). These multi-scale problems can be very difficult to solve on long time and system scales. For instance, the computational complexity of a typical plasma problem, a 2D island coalescence,

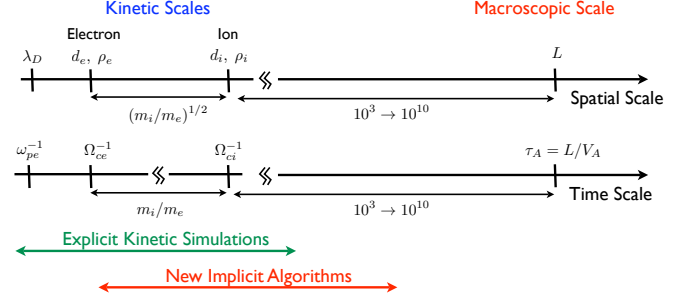


Figure 3. Kinetic Plasma Scales. Where  $\lambda_D$  is the debye length,  $\rho_e$  and  $\rho_i$  are the electron and ion gyro-radii,  $\omega_{pe}$  is the electron and ion plasma frequencies,  $\Omega_{ce}$  and  $\Omega_{ci}$  are the electron and ion plasma frequencies, and  $\tau_A$  is the Alfvén wave time scale.

scales roughly by as [11]:

$$RunTime \approx \mathcal{O} \left( \overbrace{500 \times \left( \frac{L}{\lambda_D} \right)^2}^{N \text{ Particles}} \times \overbrace{(3 \times 10^4) \frac{m_i}{m_e}}^{N \text{ time steps}} \right) \quad (1)$$

where  $L$  is the system length,  $\lambda_D$  is the Debye length<sup>2</sup>,  $m_i$  and  $m_e$  are the ion and electron masses respectively. A solution for this problem has been obtained with an explicit algorithm by Karimabadi et. al. [10] that required 500 particles per cell on a mesh of  $17920 \times 8960$  cells, and a much reduced mass ratio  $m_i/m_e = 25$  (realistic  $m_i/m_e = 1836$ ). Essentially this problem required on the order of  $10^{16}$  (number of particles  $\times$  number of time steps), total particle steps with a system-wide communication required every  $8 \times 10^{10}$  explicit particle time steps. As discussed earlier our proposed algorithmic solution reduces the difficulty of solving these multi-scale problems by breaking the problem into a coupled HO/LO solution: a coarse (macroscopic LO) solution to resolve large scale features and a fine (microscopic HO) solution to deal with the small scale features. We developed this solution using a unique co-design process.

### D. Assembling the Co-Design Process

The co-design process itself is expected to evolve as we work through a range of domain science problems, the plasma physics application is the first. The basic process we are evolving to create a multi-scale plasma physics application, shown in Figure 2, begins with identification of desired physics capabilities and architecture capabilities. Algorithms are chosen and designed to best support these two factors. A prototype-app is then used to implement and validate the scientific characteristics of the concept algorithm. This application is generally developed in a higher-level language, such as MATLAB. Once the validity of the algorithms has been proved, mini-apps are developed to test the skeleton algorithm on different architectures.

<sup>2</sup>The Debye length is the scale over which mobile charge carriers in a plasma screen out electric fields

During the development of these mini-apps, multiple micro-benchmarks are used to rapidly test various design decisions for optimizing the mini-apps. Once the simplified versions of the algorithms have been thoroughly explored, the process is repeated for more complex versions of the application. This is a co-design iteration with lessons learned from the previous iteration impacting the next iteration. Once the target problem complexity has been reached, the lessons learned from the optimized mini-apps and prototype-apps are used to develop the final compact-app.

To support our co-design process, we are developing a flexible framework for testing optimization strategies and algorithms for multiple architectures and physics problems. This framework will serve as the backbone for the compact-app paradigm shift demonstration.

This paper will focus on the development of the plasma application framework, and the optimization of its multi-core component for both 1D1V and 2D3V problems<sup>3</sup>. To begin, we review the Particle-in-Cell (PIC) algorithm, and indicate how our PIC implementation differs from standard PIC approaches.

## II. THE APPLICATION AND ALGORITHM

Particle-in-Cell (PIC) algorithms are widely used to model kinetic behaviors in plasmas. Standard PIC algorithms are explicit, and employ a leap-frog time integration scheme [2]. Explicit PIC codes are straightforward to parallelize since every particle can be integrated independently of the others. The discrete particle nature of PIC codes makes them ideal for use with fine grained parallelism accelerators such as GPUs and MIC co-processors.

There has been significant effort in developing explicit PIC codes for advanced architectures. VPIC [3], an explicit PIC code used extensively at LANL, was successfully implemented on the Roadrunner system, a hybrid AMD64 and IBM PowerPC architecture [3]. GPU based explicit PIC codes include SCEPTIC3DGPU [15], PICongPU [4], and OSIRIS [12].

While particle pushing in explicit methods maps well to advanced, highly parallel architectures, there are significant communication costs that are incurred every time step. These communication costs include gathering the charge and current tallies, and then scattering the updated field information back to all nodes. In serial applications, these communication costs are negligible compared to pushing the particles. However, once the pushing performance increases by several orders of magnitude, and the problem is spread across thousands of nodes, communication costs become the bottleneck. The key to maximizing hardware utilization is to increase the amount of particle-push work per time step in the service of numerical accuracy. We achieve this via a consistent HO/LO moment-based implicit PIC method.

<sup>3</sup>1D1V means only one spatial dimension and one velocity dimension are used.

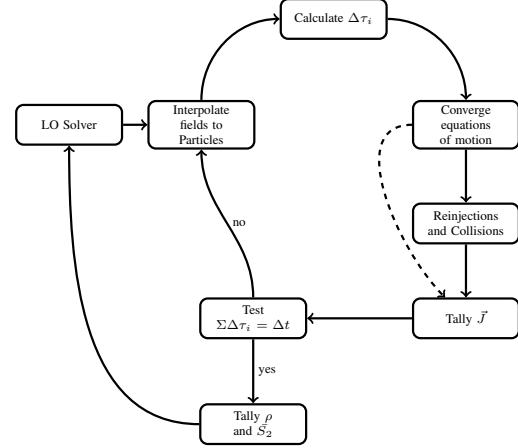


Figure 4. Implicit PIC flowchart. Starting with an initial particle distribution, the LO Solver solves the field and moment equations. The solution to the field equations are passed to the HO solver, which pushes the particles multiple subcycles. Each particle uses a different subcycle length,  $\Delta\tau$  for each subcycle. After a particle's equations of motion have been fully converged the subcycle averaged current is tallied, and in future codes reinjections and collisions will be handled. Once the particle has been pushed the full time step the density and stress tensor are tallied. After all particles have been pushed the HO moments:  $\bar{J}(p_{i,t+1/2})$ ,  $\rho(p_{i,t+1})$ , and  $\bar{S}(p_{i,t+1})$  are sent to the LO solver for the next iteration.

### A. Consistent Moment-Based Implicit PIC

Explicit PIC codes must resolve all space and time scales for numerical stability [2]. Implicit PIC is a particle based kinetic model wherein the particle's equations of motion are fully converged together with Maxwell's equations. Consistent Implicit PIC removes the numerical stability constraints that confine explicit methods to very small time steps and fine meshes [5], [16]. This is a critical advantage for simulating plasma phenomena that evolve on ion time and system length scales. A basic outline of the HO/LO moment-based implicit PIC algorithm with subcycled particle stepping can be seen in Figure 4.

The method illustrated in figure 4 begins with the initialization of the particles and High-Order (HO) moments, current  $\bar{J}(p_{i,t+1/2})$ , density  $\rho(p_{i,t+1})$ , and stress tensor  $\bar{S}(p_{i,t+1})$ . These HO moments are used to solve for the initial field values and generate the initial Low-Order (LO) moments. At this point the main time step,  $\Delta t$ , loop begins. Inside the time step loop, all particles are iterated over and pushed within a subcycle loop using timesteps  $\Delta\tau_\nu$ , where  $\Delta t = \sum_\nu \Delta\tau_\nu$ . The subcycle loop begins with the interpolation of the field values to the particle. These field values are then used to estimate  $\Delta\tau_\nu$  for that subcycle step. The particle's equations of motion are then converged using a nonlinear Picard method. Once a particle's equations of motion have been fully converged the particle's contribution to the current density  $\bar{J}$  is weighted and tallied by  $\bar{J}_{i+1/2} = \bar{J}_{i+1/2} + \frac{\Delta\tau_\nu}{\Delta t} \vec{v}$ . After a particle has completed all subcycles, the contributions to charge  $\rho_{i+1}$  and stress tensor

$\bar{S}_{i+1}$  are tallied. When all particles have been pushed the HO moments for the next time step;  $\bar{J}_{i+1/2}$ ,  $\rho_{i+1}$ , and  $\bar{S}_{i+1}$ , are used to update the LO moments and solve for the field values at the next time step.

---

**Algorithm 1** Moment-Based Implicit PIC with subcycling

---

```

for  $t = 0 \rightarrow t_{max}$  do
  while  $\text{resid} > \text{tol}$  do // Outer Picard Loop
    for all  $p_{i,0} \in \mathcal{P}_t$  do // Particle Loop
      while  $\sum \Delta\tau_\nu < \Delta t$  do // Subcycle Loop
        Interpolate  $\vec{E}_{t+1/2}, \vec{B}_{t+1/2}$  to  $p_{i,\nu}$ 
        Estimate  $\Delta\tau_\nu$ 
        Solve equations of motions:  $p_{i,\nu} \rightarrow p_{i,\nu+1}$ 
        Tally  $\vec{J}(p_{i,\nu+1/2})$ 
      end while
      Tally  $\rho(p_{i,t+1}), \bar{S}(p_{i,t+1})$ 
    end for
    Calculate residual
     $\vec{E}_{t+1}, \vec{B}_{t+1} = \text{LO}(\vec{J}(p_{i,t+1/2}), \rho(p_{i,t+1}), \bar{S}(p_{i,t+1}))$ 
  end while
end for

```

---

An important advantage of non-linearly coupling the fluid and kinetic models lies in strict charge and energy conservation theorems [5]. Charge conservation is achieved through a specific discretization scheme and forcing particles to stop at cell boundaries for current accumulation. Energy conservation depends on fully converging particles and fields, and require a specific current accumulation to the mesh and field scatter to the particles. These constraints impart significant design challenges to the development of optimized particle-orbit integration routines.

### B. Benefits of Consistent Moment-Based Implicit PIC

Consistent HO/LO moment-based implicit PIC has several architectural and algorithmic benefits. The first major benefit is that the vast majority of the pushing work is isolated at the processing unit level. This means that a very large amount of work is done per byte communicated, be it inter-node, accelerator-CPU, or multi-core communication. This is a result of the ability of the moment-based PIC method to take much larger time steps between field updates, on the order of 10-100x explicit PIC time steps. The actual particle orbit integration still takes place with smaller time step sizes, but those are trivially parallel tasks.

In 1D and 2D we are employing a partial replication approach, which replicates all of the moments and fields on every node, but not the particles. The total inter-node communication for this approach is contained within a single `MPI_allreduce()` that is called after all particle subcycles have been completed. This approach, combined with the very large amount of on node work required to push the particles, results in the minimal contribution of communication to overall computation time. This effect can

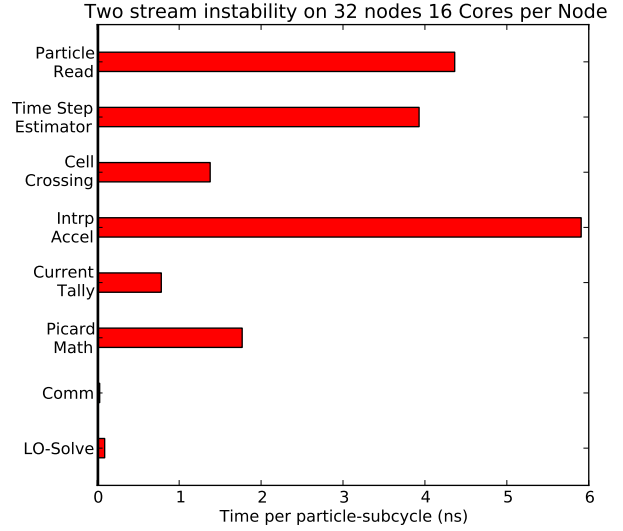


Figure 5. Multi-Node 1D1V Two Stream Instability Results.

be seen in Figure 5, which has been obtained with a 1D1V electrostatic two-stream instability simulation on 32 nodes with 16 cores per node. The top six bars represent the HO solver effort while the bottom bar represents the LO solver effort. On-node parallelization is done using OpenMP, while inter-node parallelization is done using MPI. Essentially communication costs are non-existent next to the pushing costs.

The second major architectural benefit of this method is the fact that it maps well to heterogeneous architectures. Pushing particles is a trivially parallel task that is well suited to GPUs and other accelerators, such as MIC. Since a very large amount of work can be isolated on the accelerator, the high latency and low bandwidth limits of the PCIe bus are largely irrelevant. Additionally, since some problems deal with multiple particle species, which require a varying amount of work to push, different hardware can push different species and numbers of particles in order to attain load balancing. For example we push ions and a small number of electrons on the CPUs and electrons only on the GPUs since the electrons require a greater number of subcycles due to lower mass.

In addition to the architectural benefits, as previously stated, consistent implicit PIC offers substantial accuracy benefits such as strict charge and energy conservation over large time scales. Enforcing the conservation of these properties limits noise modes to only those that are both charge and energy conserving, and removes uncertainty that can arise from numerical heating and cooling. Cummins et. al. has demonstrated that near energy-conserving implicit models can substantially reduce noise in material simulations [7], and we are currently working on demonstrating similar

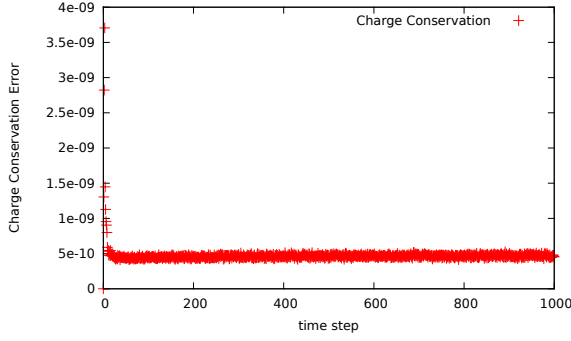


Figure 6. 2D3V Charge Conservation for 1000 timesteps of  $10\omega_{pe}^{-1}$  each (roughly 100,000 explicit time steps).

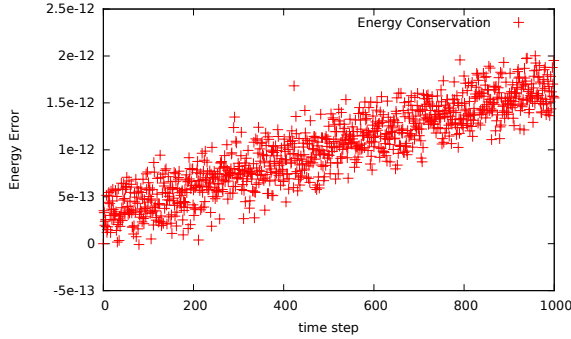


Figure 7. 2D3V Energy Conservation for 1000 timesteps of  $10\omega_{pe}^{-1}$  each (roughly 100,000 explicit time steps).

benefits in plasma physics problems. Both properties are conserved roughly to floating point tolerance. Figures 6 and 7 demonstrate conservation of both properties over 1000 time steps, with a time step size of  $10\omega_{pe}^{-1}$  (roughly 100,000 explicit time steps) for a 2D3V magnetic Island Coalescence equilibrium problem. Energy conservation requires specific discretizations for the field interpolations and moment tallies. Charge conservation requires particles to stop at cell faces and tally the half time current at the cell face [6]. Stopping at the cell face requires a specialized cell crossing algorithm, which will be discussed in a future manuscript.

### III. CO-DESIGN EFFORT

The plasma app has achieved significant progress in a short amount of time in part due to previous experience inherited from previous projects [5], [6], [15]. These previous experiences include 1D1V electrostatic implicit-particle-in-cell codes implemented on both the CPU and GPU, detailed in [5], [6], and a GPU implementation of the 3D3V explicit particle-in-cell code SCEPTIC3D, detailed in [15].

The charge and energy conserving implicit PIC algorithm was put forth by Chen and Chacón [5]. Taitano [16] coupled this implicit PIC scheme with a moment-based acceleration which allowed for effective nonlinear convergence within a time-step without the use of the outer Newton-Krylov

solver in [5]. Both Chen and Taitano separately developed prototype-apps to test the validity of the HO-LO moment based, charge and energy conserving, implicit PIC algorithm for 1D1V, electrostatic problems [5], [16]. Following the physics demonstration, a 1D1V electrostatic mini-app was developed to investigate the HO portion algorithm on GPUs [6].

The current effort is focused on developing a flexible parallel PIC framework, which incorporates multiple physics problems, spanning different spatial and velocity dimensionalities, and architecture specific implementations and optimizations. This framework is essentially the foundation for multiple prototype and mini apps and is designed such that the major components are essentially plug and play. A TRILINOS [8] based LO solver has been incorporated into the framework and currently solves 1D1V and 1D3V problems. Different mini-apps can be generated simply by using different combinations of LO solvers, field storage and access strategies, and particle data storage strategies.

#### A. PlasmaApp

PlasmaApp is a flexible multi-architecture implicit particle-in-cell framework that handles problems of varying spatial and velocity dimensionality. This PIC code differs from all other PIC codes in that it is non-linearly implicit, and is both charge and energy conserving.

The goals of the PlasmaApp framework are as follows:

- Only implement the physics once, for all architectures.
- Support multiple architectures, multi-core, GPU, MIC.
- Support multiple optimization paths for each architecture.
- Parallelize across multiple nodes.

1) *Physics Abstraction*: The first guiding philosophy in the development of this framework is the concept of a single physics implementation. For each architecture the domain is exactly the same, The specific instructions used to perform that math, and the data management systems used to supply the information to be operated on may be different, but at some higher level, the operations performed by each device should appear almost identical. This higher level is where the physics should be implemented.

For the purposes of this framework, much of the common physics is contained in the particle orbit integration. This section of the framework is laid out in such a way that an application scientist can immediately recognize the physics being performed, but also in such a way that the compiler can generate high performance architecture specific implementations. In C++ the primary mechanisms for telling the compiler to generate different versions of the same code are templates and preprocessor statements. The solution used in this framework is a strongly templated particle-physics class, that encapsulates the particle orbit integration.

An important consideration for this physics object is that the compiler should generate different code for different

spatial and velocity dimensionalities, as well as electrostatic vs electromagnetic models. This allows the framework to choose the appropriate particle physics for the problem.

2) *Operation Abstraction*: Another consideration in designing this physics object is that of the SIMD nature of the target devices. For multi-core and MIC architectures the vector processing is done in a  $1 \times N$ -wide vector, that is one  $N$ -wide vector is processed per thread at a time. The GPU handling is essentially the transpose, handling  $N \times 1$ -wide vectors per multi-processor, basically a 1-wide vector per thread. This means that the physics object should be templated such that it can be used as a structure of arrays or an array of structures. An example of this object is shown below:

```
template<int N,int nS,int nV,bool iEM>
class ParticleObjNT
{
private:
    // position within cell range [0:1)
    typevecN<typevecN<double,N>,nS> position;
    // particle velocities
    typevecN<typevecN<double,N>,nV> velocity;
    // cell index
    typevecN<typevecN<int,N>,nS> iposition;
};
```

This templated particle object can act as either an array of structures for vectorization on CPU or MIC architectures, or as a structure of arrays on GPUs.  $N$  is the number of particles handled by the object,  $nS$  is the number of spatial dimensions, and  $nV$  is the number of velocity dimensions.

Vectorization and device specific intrinsic operations are hidden within the `typevecN<T,N>` object where  $T$  is the data type, and  $N$  is the length of the vector. All of the relevant mathematical operations are overloaded for this data type to loop over all elements within the vector. This allows most of the mathematical operations performed in the code to be expressed in simple for-loops, which are easily vectorizable by the compiler.

Additionally, all of these operations can be inlined, and set to architecture specific operations through the use of preprocessor directives. An example of this operator overloading is shown below:

```
template<typename T, const int N>
typevecN<T,N> __fmad(typevecN<T,N>& a,
                    typevecN<T,N>& b,
                    typevecN<T,N>& c)
{
    typevecN<T,N> d;
    for(int i=0;i<N;i++) {
#ifdef CUDA_CODE
        d(i) = __fma_rn(a(i),b(i),c(i));
#else
        d(i) = a(i) * b(i) + c(i);
#endif
    }
    return d;
}
```

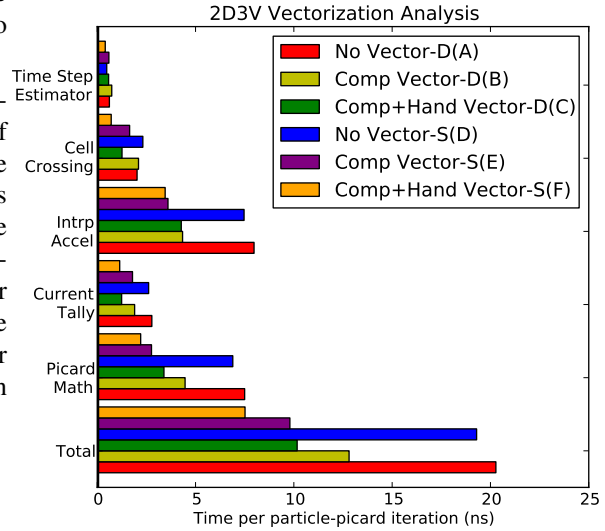


Figure 8. Single Node vectorization analysis.

Overloading operators in this manner enables the problem to be laid out in such a way that the compiler can easily understand the loops and vectorize them. Additionally it allows for easy integration of hand vectorization. In order to analyze the importance of vectorization we ran three tests with differing levels of vectorization for both single and double precision. These tests were performed on an Intel Sandy Bridge CPU with support for 256-bit AVX instructions. The results of these tests are shown in figure 8.

Cases (A) and (D) were run with a pseudo-vector length of 1, with all compiler -O3 flags on, minus the compiler vectorization. An interesting feature of these tests is that the single precision performance is almost identical to the double. This is due to the fact that the majority of floating-point operations in the SSE instruction set have roughly the same latency and throughput for both double and single precision [1].

Cases (B) and (E) test the ability of the compiler to vectorize the code. These tests were run with all -O3 flags on and a pseudo-vector length of 32. It is interesting to note that the compiler vectorization does well in most of the routines, except for the cell-crossing and time step estimator. Here the single precision version is faster than the double due to the fact that twice the number of operations are performed for the single versions compared to the double.

The last cases, (C) and (F), incorporate routines that were hand vectorized using AVX intrinsics in addition to the compiler vectorized code. The hand vectorized operators were min, max, floor, signum, integer modulo, and abs. This optimization had a significant effect on the cell-crossing, current tally, and general math, but little impact on

the acceleration interpolation and time step estimation.

Overall, the effects of compiler vectorization and hand vectorization resulted in roughly a 3x speedup for the single precision case, and 2x for double precision for 256-bit wide vectors. While a factor of 2x speed up is not particularly impressive, it does demonstrate that we are able to achieve a high degree of vectorization. We expect the performance gap to grow further as CPU architectures move from high serial performance to more parallel performance, i.e. wider vectors such as those on Intel MIC and AVX2 enabled processors.

3) *Optimization and Hardware Abstraction:* The third consideration concerns the flexibility to support different data management strategies for different architectures and optimizations. Our approach is to use virtual classes with optimization and architecture specific instantiations. For the purposes of this application, the data structures that will require architecture specific implementations are the particle information arrays, the field information, and how the moments are tallied. Fortunately, both CUDA and MIC support virtual classes, although with some caveats for CUDA.

### B. Analysis and Optimization Process

The PlasmaApp framework is designed with various levels of abstraction in order to facilitate our analysis and optimization process. For the purposes of the multi-core mini-app this process is as follows:

- 1) Run performance benchmarks to determine most costly routines.
- 2) Use tools to determine if the costs are math or memory related.
- 3) Identify strategies for improving the performance of costly routines.
- 4) Implement different optimization strategies in an easily reversible manner.

### C. Process Example: Multi-core 2D3V Optimization

Before delving deeply into optimizing for advanced architectures we first produced an optimized multi-core Xeon version of the PlasmaApp framework. To this end, two tools were employed: a high-performance in-code timer and an LLVM-based profiler called Byfl [14]. Byfl is a tool developed at LANL with the goal of helping application developers understand code performance in a *hardware-independent* way by instrumenting code in LLVM's intermediate representation.

1) *Profiling Analysis:* The first step in the optimization and analysis process involves profiling the main components of the code and determining the cost breakdown. This is done using a high resolution, low overhead timer based on the *rdtsc* instruction. Of course, this method is only reliable when the core clock speed is kept constant. Measurements of the relative error between the *rdtsc* instruction and *clock\_gettime()* were in the range of  $10^{-6}$ . These timers wrap each of the main routines in the PlasmaApp

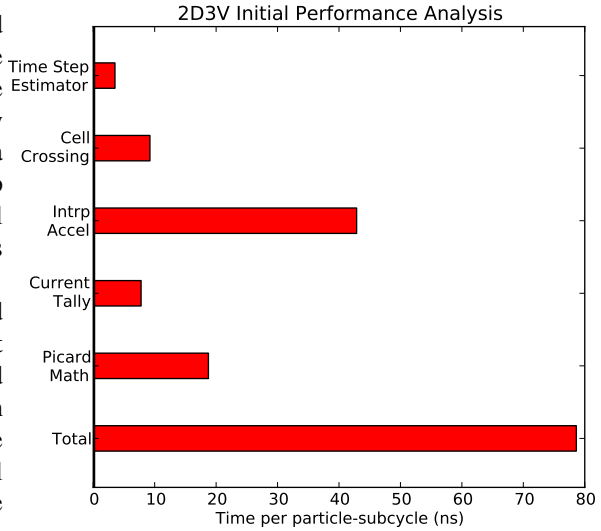


Figure 9. Initial performance profiling of 2D3V problem,  $7.17 \times 10^8$  subcycles.

code. The total runtime from each timer is then divided by the total number of particle subcycles. The profiling results for the 2D3V island coalescence equilibrium problem with 512,000 particles on a 32x32 mesh, on a machine with 2x Intel Xeon E5-2687W 8-core CPUs at 3.1GHz are shown in figure 9.

2) *Byfl Analysis:* The acceleration interpolation was previously identified as the most costly routine. Using the Byfl [14] tool, we measure that each particle requires roughly 2300 floating point operations per subcycle. The acceleration interpolation alone contributes 1,500 flops per subcycle, or about 64%. Specific, dominant, functions which are being called numerous times per subcycle have been identified. These include *fabs()*, *fmin()*, and the field interpolation functions.

3) *Possible Optimization Strategies:* The main costs identified were related to memory operation efficiency in the acceleration interpolation and the evaluation of various mathematical operations. Based on this information, and knowledge of the underlying algorithm, we can identify several possible optimization strategies: 1) reduce the number of floating point operations, 2) increase the number of bits used per memory op.

First we can hand vectorize frequently evaluated math functions that are not being taken care of by the compiler, such as *min*, *max*, *abs*, *floor*, *mod*, and *signum*. This will serve to speed up much of the general Picard math, and a vectorized *signum* should speed up the cell-crossing algorithm.

Second, we can consolidate the shape function evaluations in the acceleration interpolation, since some shape functions are used multiple times in a given acceleration interpolation.

This will serve to significantly reduce the total number of shape function evaluations required each time step. Hand vectorizing the interpolation functions can also be investigated using a micro-benchmark.

Lastly we can try different field data storage strategies. The original implementation used a structure of arrays approach, it is possible that an array of structures approach might be better.

Micro-benchmarks can be developed to test some of these optimization ideas. One of the micro-benchmarks used for this optimization phase was an AVX implementation of the shape function calls for field to particle interpolation.

4) *Hand Vectorization of Shape Functions.*: During the interpolation of field information to particles, the field values at various mesh points are weighted using of specific shape functions. The order of the shape functions used depends on the location and component of the field value being interpolated. In the case of a 2D3V problem first and second order shape functions are used. These shape functions are shown in Eq. 2 and 3 below, where  $x$  is the normalized distance between a particle and a grid point.

$$S_1(x) = \begin{cases} 1 - |x| & \text{for } |x| \leq 1 \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

$$S_2(x) = \begin{cases} \frac{3}{4} - |x|^2 & \text{for } |x| \leq \frac{1}{2} \\ \frac{1}{2} \left( \frac{3}{2} - |x| \right)^2 & \text{for } \frac{1}{2} \leq |x| \leq \frac{3}{2} \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

The shape function micro benchmark involves computing four S1 values and six S2 values, the same number of evaluations as in the 2D3V electromagnetic field interpolation. These 10 evaluations are performed many times in order to reduce variation in the results. Using the approach we tested both double and single precision 256-bit AVX implementations of the shape function evaluations and compared the run times to scalar versions. The single precision AVX implementation was 2x faster than the scalar float version, while the double precision AVX version was only 63% faster than the scalar double version.

5) *Results of Optimization Strategies.*: In the end, we implemented three different optimizations, and analyzed how they interact with each other in five different tests (see Figure 10). The first test case (A), Opt=000, is the baseline performance result. Case (B), Opt=100, consolidates the shape function evaluations. Case (C), Opt=110, both consolidates the shape function evaluations and stores the electric and magnetic fields as an array of structures. Case (D), Opt=001, includes hand vectorized routines. Case (E), Opt=101, consolidates shape function evaluations, and employs hand vectorized routines, including the hand-vectorized shape function evaluations. The final case, case (F), Opt=111, includes all of the optimizations.

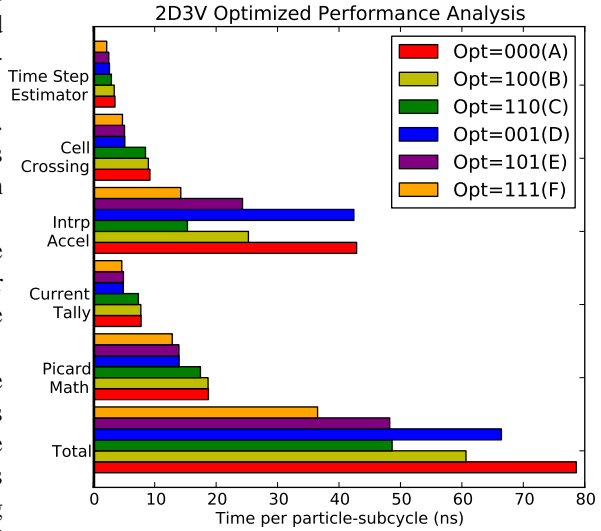


Figure 10. 2D3V Island Coalescence Equilibrium optimization results (double precision).

As is shown in Figure 10, consolidating the shape function evaluations reduces the cost of the acceleration interpolation by nearly 2x. The exact cause of this speedup can be seen in a Byfl analysis of the consolidation optimization. Prior to the optimization, the acceleration interpolation comprised 1500 out of the total 2300 flops per subcycle. Consolidating the shape function evaluations reduced that to 700 out of the total 1500 flops per subcycle.

Storing the field data as an array of structures instead of a structure of arrays also provided a sizable performance boost. Hand vectorizing several additional functions did not provide much of a performance boost in any given routine, but together all three optimizations account for an overall factor of 2x speedup for the double precision version of the code.

#### IV. CONCLUSIONS

As computing platforms become increasingly heterogeneous new algorithms and applications are needed in order to take advantage of these systems. Applications and algorithms must adapt to the new architectures, while programming models and abstractions must adapt to the new applications and algorithms.

In this paper we described a co-design process used to facilitate these adaptations through the use of new algorithms, abstraction layers, and proxy-applications. We are applying this process to multi-scale plasma kinetic simulation. The hierarchical nature of the new HO-LO moment-based implicit PIC algorithms developed for this simulation maps well to the nature of the emerging architectures. Proxy applications were used to develop and test the validity of the new algorithm, as well as to develop new abstraction layers

for hiding problem dimensionality and architecture specific implementations. We have demonstrated that a high degree of performance can be achieved on multi-core systems with the abstraction methods presented, and we do not foresee these abstraction methods inhibiting performance on other architectures. We also do not foresee scalability becoming an issue simply due to the significant amount of work performed per communication for our problems.

The next steps will be to develop and implement the GPU and MIC components of the PlasmaApp framework. Work is currently being done to develop the LO solver components for both 1D3V and 2D3V electromagnetic problems. Mapped mesh support is also planned, and will be implemented along with the GPU and MIC components.

#### REFERENCES

- [1] Intel® 64 and IA-32 Architectures Optimization Reference Manual. Technical report, Intel, Apr 2012.
- [2] C. Birdsall. *Plasma physics via computer simulation*. McGraw-Hill, New York, 1985.
- [3] K. J. Bowers, B. J. Albright, L. Yin, W. Daughton, V. Roytershteyn, B. Bergen, and T. J. T. Kwan. Advances in petascale kinetic plasma simulation with vplic and roadrunner. *Journal of Physics: Conference Series*, 180(1):012055, 2009.
- [4] H. Burau, R. Widera, W. Honig, G. Juckeland, A. Debus, T. Kluge, U. Schramm, T. Cowan, R. Sauerbrey, and M. Bussmann. PIONGPU: A fully relativistic particle-in-cell code for a GPU cluster. *Plasma Science, IEEE Transactions on*, 38(10):2831–2839, 2010.
- [5] G. Chen, L. Chacón, and D. C. Barnes. An energy- and charge-conserving, implicit, electrostatic particle-in-cell algorithm. *Journal of Computational Physics*, 230:7018–7036, Aug. 2011.
- [6] G. Chen, L. Chacón, and D. C. Barnes. An efficient mixed-precision, hybrid CPU-GPU implementation of a nonlinearly implicit one-dimensional particle-in-cell algorithm. *Journal of Computational Physics*, 231(16):5374 – 5388, 2012.
- [7] S. Cummins and J. Brackbill. An implicit particle-in-cell method for granular materials. *Journal of Computational Physics*, 180(2):506–548, 2002.
- [8] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the Trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423, 2005.
- [9] R. Hockney and J. Eastwood. *Computer Simulation Using Particles*. Taylor & Francis, 2010.
- [10] H. Karimabadi, J. Dorelli, V. Roytershteyn, W. Daughton, and L. Chacón. Flux pileup in collisionless magnetic reconnection: Bursty interaction of large flux ropes. *Phys. Rev. Lett.*, 107:025002, Jul 2011.
- [11] D. A. Knoll. Plasma paradigm shift document for the CoCoMANS project. Technical report, Los Alamos National Laboratory, Mar 2013. LA-UR-13-22126.
- [12] X. Kong, M. C. Huang, and C. Ren. Preliminary results on GPU acceleration of the PIC simulation code OSIRIS using CUDA. Presented at the 51st Annual Meeting of the APS Division of Plasma Physics, 2009.
- [13] A. Munshi et al. The OpenCL specification. *Khronos OpenCL Working Group*, 1:11–15, 2009.
- [14] S. Pakin. Compiler-based application analysis. Exascale Research Conference, April 2012.
- [15] J. E. Payne. Implementation and performance evaluation of a GPU particle-in-cell code, 2012.
- [16] W. Taitano, D. A. Knoll, L. Chacón, and G. Chen. Development of consistent and stable fully implicit moment method for Vlasov-Ampere particle in cell (PIC) system. *SIAM J. Sci. Comput.*, 2013. accepted.